Resource support

Srini, rs@toprllc.com

September 6, 2020

1 Introduction

This projectlet enables applications to include reources such as graphics files, audio files in the executable for distribution; instead of as individual files. This is achieved by converting the files into source code to be compiled for inclusion in the binaries. At runtime, the application can retrieve the resource with a runtime library.

- Generated sources can be in one of several languages : **C, Ada, go**.
- A hash of the file is generated and included in the binary for runtime verification. This will protect the binary from corruption and/or tampering.
- Project configuration is provided in **json** format.
- The utility can implement a simplistic encryption of the data before generating the output. This is to enable incorporation of API keys and the like into the binary.The goal is to make it a little harder to extract the secrets.

1.1 Learning Objectives

- Process json formatted files. **json** is ubiquitous as the vehicle for data exchange. Most programming languages support json processing without each of us having to develop it from scratch. Choosing an appropriate library is part of this exercise.
- Polymorphism - Given a configuration in json, this projectlet converts files into source code compilable for inclusion in an executable. Depending on the user the preference may be for a variety of programming languages. Your programming language might enable this in some way e.g. **base class** and **derived** classes in **C++** while for **go** the technique might be built on the concept of **interface**s.
- In order to protect against corruption of the data even if included in a binary, we could several techniques. A simple way might be to calculate a checksum or hash and verify the value at runtime before using the data. Here again, an appropriate library has to be chosen instead of implementing the algorithm.
- Unit testing is an important aspect of software engineering. Though there seem to be numerous frameworks for unit testing, it does not have to be too fanciful. The project can be divided into smaller modules e.g. command line processing, loading the configuration data from the file and the actual conversion of files into resources; unit tests can be developed for each module.

- Exploration of a simplistic approach to data encryption/decryption.

1.2 Configuration

A sample configuration for a project file is listed below.

Listing 1: Example config.json

```
{
    "resources" : [
        {
            "name" : "RESOURCE1" ,
            "filename" : "res1.png"
        } ,
        {
            "name" : "RESOURCE2" ,
            "filename" : "source.h"
        }
    ]
}
```

2 Implementation

An example in **C++** is developed using the following external libraries:

Libraries

- JSON processing is achieved using the **libjansson** (`https://digip.org/jansson/`). This library is a stable library in **C** - quite simple to integrate.
- Cryptographic hash of files is generated using the MD5 algorithm. The library GTK3 from the gnome project (`https://developer.gnome.org/references`) is used for this purpose.

Encryption The fundamental building block used is the **xor** algorithm. In order to keep the password management simple, the creation date of the output structure is used as the key for this algorithm. The input is processed in blocks of a configurable size and the timestamp is repeated as many times as needed to fill up the encryption key of size equal to this block length.

Exactly identical algorithm is performed by the runtime to decrypt the data. The required parameters namely the creation date and the blocklength are to be conveyed to the runtime - which is added to the output files - a header file in the case of C and a package spec in the case of Ada.

Implementation in **C++** : `https://gitlab.com/cpp8/bindata.git`

2.1 Potential enhancements

- Text files can be potentially decoded from the hexadecimal conversions. An alternate representation might make it a little harder.

- Some files such as text files may benefit from compression. The compressed data can become the compilable source.